Universidad Autónoma de Madrid

Escuela Politécnica Superior

Proyecto para la obtención del título de
Máster en Investigación e Innovación en
Inteligencia Computacional y Sistemas Interactivos
por la Universidad Autónoma de Madrid

Tutor del trabajo fin de máster:
Gonzalo Martínez Muñoz

REDOL: Conjunto Estocástico de Clasificadores Basado en la Distinción de las Etiquetas de Salida

Mario Calle Romero

Mario Calle Romero
REDOL: Conjunto Estocástico de Clasificadores Basado en la Distinción de las Etiquetas de Salida

Mario Calle Romero
Avenida\ Navarrondán, Nº 19

IMPRESO EN ESPAÑA – PRINTED IN SPAIN

ÍNDICE

Listas

Lista de figuras

Resumen

En este trabajo se propone un novedoso modelo de clasificación basado en conjuntos de clasificadores. El método propuesto transforma el paradigma tradicional del reconocimiento de patrones en el que una clase es asignada a cada instancia de los datos, por una nueva idea en la que el modelo es entrenado para diferenciar si una instancia está correctamente etiquetada. Para ello, cada clasificador combinado en el conjunto se entrena sobre un conjunto de datos diferente en el que varias etiquetas de las instancias son modificadas. Las clases modificadas son añadidas como una nueva etiqueta del conjunto de datos de entrenamiento, y las nuevas etiquetas de este conjunto describen si la instancia está "correcta" o "incorrectamente" clasificada. El modelo es analizado con una serie de experimentos sobre 15 conjuntos de datos diferentes. El propósito de estos experimentos es estudiar la evolución en el error de generalización, la frontera de decisión que consigue el modelo, la diversidad del clasificador y el comportamiento de sus hiperparámetros, así como validar su rendimiento con respecto a algunos de los conjuntos de clasificadores más utilizados como Random Forest o Gradient Boosting.

Se propone un amplio abanico de opciones optimizables para el método propuesto que ayudan a resolver los problemas analizados. El método se ha evaluado sobre una batería de experimentos en los que se ha podido comprobar que el modelo implementado tiende a la convergencia del error de generalización a medida que aumenta el número de clasificadores base combinados, demuestra resultados favorables sin realizar optimización de hiperparámetros en comparación a otros algoritmos basados en conjuntos de clasificadores y obtiene errores de generalización más bajos al optimizar los hiperparámetros propuestos que el algoritmo Random Forest. El método propuesto ha demostrado resultados remarcables en los experimentos propuestos en este proyecto, obteniendo los mejores resultados entre los errores de generalización en comparación con el resto de algoritmos que definen el estado del arte en el *ensemble learning*.

Abstract

This project proposes a novel classification model based on ensemble learning. The proposed method transforms the traditional pattern recognition paradigm in which for each instance a new label is assigned, into a new idea in which the model is trained to distinguish whether an instance is correctly labeled. To do so, each classifier combined in the ensemble is trained on a different dataset in which several instances' targets are modified. The modified classes are added as a completely new feature into the training dataset and the new labels are expressed as if an instance is correctly classified or not. The model is analyzed with a series of experiments on 15 different datasets. The purpose of these experiments is to show the generalization error evolution, the decision border formed, the classifier diversity and its hyperparameters behavior as long as validate its performance with respect to some of the most commonly used ensemble classifiers such as Random Forest or Gradient Boosting.

The proposed method has shown an incredibly wide range of options in order to solve some of the tasks analyzed. The method has been evaluated over several experiments in which it has been found that the implemented model tends to converge as the number of base classifiers is increased, it shows great results when hyperparameters are not optimized in comparison to state-of-the-art ensemble learning models and it obtains lower generalization errors compared to Random Forest when hyperparameters are optimized. The novel method has shown remarkable results in the experiments proposed in this project, obtaining some of the best results among the generalization error comparison with the rest of the ensemble learning classifiers.

1

INTRODUCCIÓN

1.1. Motivación

Durante los últimos años se han hecho grandes avances en distintas áreas de la inteligencia artificial por su gran relevancia a la hora de resolver problemas fácilmente automatizables y la transversalidad de áreas que se pueden aplicar las técnicas de aprendizaje automático. El aprendizaje automático, concretamente, puede ser una herramienta de apoyo en áreas como la sanidad (Nemesure et al., 2021; Ravaut et al., 2021) o la educación (Gupta and Batra, 2021) entre otras. Una de las áreas que más se ha estudiado del aprendizaje automático es el reconocimiento de patrones o clasificación. Reconocimiento de imágenes o de texto son tareas en las que se ha trabajado y progresado mucho durante la última década mediante el uso de redes profundas convolucionales (Aggarwal and Zhai, 2012; Alzubaidi et al., 2021; Smith et al., 2021). En cualquier caso, el reconocimiento de patrones sigue estando muy extendido en la industria, para problemas con datos tabulares –pues su presencia sigue siendo principal en el día a día de las personas y las empresas– dónde destacan multitud de algoritmos. Dentro de los algoritmos de aprendizaje automático para datos tabulares podemos destacar los modelos basados en la combinación de clasificadores más simples como pueden ser los algoritmos Random Forest, Bagging o modelos basados en Boosting. Este tipo de métodos ha demostrado una gran eficacia en la resolución de problemas cuyos conjuntos de datos son tabulares (Zhang et al., 2017).

Los modelos de reconocimiento de patrones basados en conjuntos de clasificadores (o *ensemble learning*) varían tanto en la estructura de estos como en la metodología que utilizan. Por ejemplo, los algoritmos basados en Boosting (Freund and Schapire, 1997; Friedman, 2001; Prokhorenkova et al., 2018) dan un mayor peso a aquellos ejemplos o instancias que, durante el entrenamiento de los modelos sencillos, han resultado ser más complicados de distinguir. El objetivo de estos algoritmos es por lo tanto centrar sus esfuerzos en estas instancias más complicadas de clasificar. Por otro lado, los algoritmos como Bagging (Breiman, 1996b) o Random Forest (Breiman, 2001) se basan en la aleatorización, o bien de los conjuntos de datos con los que se entrenan los modelos más sencillos, o bien en la inserción de ciertas modificaciones aleatorias de los algoritmos *per se* que se combinan.

En este contexto, este proyecto describe y analiza la introducción de un novedoso modelo de apren-

dizaje automático para el reconocimiento de patrones basado en la combinación y aleatorización de modelos más sencillos como son los árboles de decisión. Además, este nuevo modelo intenta abordar el problema del reconocimiento de patrones desde una perspectiva innovadora en la que no se intenta clasificar de forma directa cada instancia, sino que el objetivo es distinguir qué instancias de los datos están bien clasificadas y cuáles están clasificadas incorrectamente.

1.2. Objetivos

Los objetivos principales de este trabajo son:

- Ampliar un modelo basado en conjuntos de clasificadores y en la aleatorización de las clases de las instancias de entrenamiento para la identificación del etiquetado de los ejemplos multiclase y ampliar las diversas opciones de aleatorización.
- Ampliar el modelo para regresión. En lugar de identificar las etiquetas, sugerir un valor de salida de regresión.
- Caracterizar de manera sistemática y sólida el funcionamiento de los algoritmos propuestos y el efecto de los distintos hiperparámetros.
- Formalizar una batería de experimentos basados en métricas de evaluación que consoliden una comparación exhaustiva entre distintos modelos basados en conjuntos de clasificadores y, así, evaluar la validez del modelo en este trabajo desarrollado.

1.3. Organización del informe

La memoria está organizada de la siguiente manera:

Capítulos 1 y 2: Resumen y Abstract. En este capítulo se explica de forma concisa, resumida y autoexplicativa el objetivo, el desarrollo y los resultados que se han obtenido durante el transcurso de este trabajo.

Capítulo 3: Introducción. En este capítulo se describe cuál es la motivación que ha llevado a cabo la realización de este trabajo y los objetivos que se pretenden conseguir.

Capítulo 4: Estado del arte. Para la explicación del algoritmo desarrollado es necesario que se expliquen previamente algunos conceptos clave que ayudan al análisis del trabajo. Estos aspectos se explican en este capítulo.

Capítulo 5: Descripción del algoritmo. Es este capítulo se explica paso a paso el proceso que sigue el modelo que se ha desarrollado para realizar el entrenamiento y la clasificación de los datos, así como todos los parámetros implicados. También se explica la aproximación que se le da a uno de los objetivos del trabajo que intenta transformar el algoritmo de clasificación en un algoritmo de regresión, así como los experimentos, métricas y resultados

que se han utilizado y obtenido para con este nuevo algoritmo.

Capítulo 6: Experimentos. Se han realizado múltiples experimentos para validar el funcionamiento del algoritmo desarrollado. En este capítulo se detallan los experimentos que se han realizado así como los resultados obtenidos y la discusión de los mismos. En este capítulo se comprueba si los objetivos del trabajo se han llevado a cabo y con qué matices se cumplen.

Capítulo 7: Resultados y discusión. En este capítulo se analizan los resultados que se han obtenidos en los experimentos realizados y se discute si se han alcanzado de forma satisfactoria los objetivos perseguidos.

Capítulo 8: Conclusión y trabajo futuro. Finalmente, en la conclusión se resume el trabajo que se ha realizado, los experimentos y los resultados obtenidos. En el trabajo futuro se plantean posibles aproximaciones futuras al modelo desarrollado.

2

Estado del arte

En la era de los datos, dónde la capacidad de almacenaje y procesamiento de las computadoras está tan avanzado que incluso en nuestros propios bolsillos podemos almacenar terabytes de información, el procesamiento, análisis y explotación de esta información ingente se ha vuelto prácticamente indispensable. La inteligencia artificial busca proporcionar a las máquinas la capacidad de aprender o identificar correlaciones estadísticas de forma automática en los datos, especialmente el área del aprendizaje automático.

El aprendizaje automático ha sido una de las áreas dónde más se ha avanzado e investigado durante los últimos años. El aprendizaje automático no sólo ha evolucionado en términos académicos, sino que su introducción en la industria ha sido inminente en las últimas décadas. Los recursos invertidos en la investigación en la inteligencia artificial se han debido al gran potencial de apoyo que supone en áreas transversales en la sociedad como pueden ser la educación, la sanidad, la economía, e incluso su intervención se ha observado imprescindible en eventos importantes de la historia como las elecciones de Estados Unidos de 2016 (Tacchini et al., 2017) o en la predicción de diagnósticos durante la pandemia de 2020 del virus COVID-19 (Wang et al., 2020).

El aprendizaje automático comprende varias ramas como el aprendizaje no supervisado, donde los datos no están etiquetados, o el área del aprendizaje supervisado, dónde destacan el reconocimiento de patrones –o clasificación– y la regresión, donde los datos están etiquetados con valores discretos o continuos, respectivamente.

2.1. Aprendizaje no supervisado

El aprendizaje no supervisado parte de conjuntos de datos no etiquetados con el objetivo de particionar o segmentar las instancias en conjuntos similares en base a las características que comparten (Preud'homme et al., 2021). Esto es, partiendo de un conjunto de datos de N instancias como $Q = \{\mathbf{x}^{(i)}\}_{i=1}^{N}$, con $\mathbf{x} \in \mathcal{X}^D$, donde $\mathbf{x}$ es un vector de D dimensiones, el objetivo de los modelos de aprendizaje no supervisado es agrupar las instancias x_i del conjunto Q de forma que si $x_i \approx x_z$ entonces $x_i, x_z \in Q_k$, siendo x_i y x_z dos instancias del conjunto de datos y Q_k un subconjunto de

instancias generado mediante técnicas de aprendizaje no supervisado. Dos instancias se consideran similares si pertenecen al mismo subconjunto de datos, y se consideran no similares si pertenecen a un subconjunto diferente. Entre las técnicas para generar estos subconjuntos de datos destacan el clustering jerárquico o la mezcla de gaussianas (Bishop, 2006).

2.2. Aprendizaje supervisado: Regresión

A diferencia del aprendizaje no supervisado, los métodos de aprendizaje supervisado tanto de regresión como de clasificación, parten de conjuntos de datos previamente etiquetados. El objetivo de los algoritmos de regresión es detectar las relaciones que existen entre los atributos del conjunto de datos para poder asignarle un valor continuo a instancias cuya etiqueta desconocemos. Es decir, partiendo de un conjunto de datos de N instancias como $Q = \{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^{N}$, donde $\mathbf{x} \in \mathcal{X}^D$ es un vector de D dimensiones e $y \in \mathbb{R}$ es un valor real, el objetivo de los modelos de regresión es encontrar la relación entre los D atributos de x y el valor $y \in \mathbb{R}$ (Bishop, 2006).

Uno de los métodos destacados para realizar regresión son los árboles de decisión. Los árboles de decisión realizan separaciones de los datos en base a reglas. Cada regla de los árboles de decisión está construida a partir de los atributos de los datos y realiza una segmentación de estos. Cada espacio generado con la segmentación engloba un valor de etiquetado de los datos (Breiman et al., 2017).

La efectividad de los modelos de regresión se puede evaluar con técnicas como Mean Absolute Error (MAE) o Root Mean Squared Error (RMSE). Estas métricas son 2.1 y 2.2, respectivamente.

$$MAE = \frac{\sum_{i=1}^{N} |y_i - \hat{y_i}|}{N} \tag{2.1}$$

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} (y_i - \hat{y_i})^2}{N}} \tag{2.2}$$

Estas métricas miden la diferencia entre los valores originales del conjunto de datos con el que se prueba, con los valores predichos por el método de regresión para calcular el error de generalización.

2.3. Aprendizaje supervisado: Clasificación

El reconocimiento de patrones o clasificación también está comprendido dentro del aprendizaje supervisado. Esto quiere decir que, al igual que la regresión, parte de un conjunto de datos previamente etiquetado. Sin embargo, en el caso de la clasificación, las etiquetas de los datos pertenecen a un con-

junto discreto de valores. Considerando un conjunto de datos de N instancias como $Q = \{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^{N}$, con $\mathbf{x} \in \mathcal{X}^D$ y $y \in \mathcal{Y}$, donde $\mathbf{x} \in \mathcal{X}^D$ es un vector de D dimensiones y $\mathcal{Y}$ es un conjunto discreto de etiquetas $\mathcal{Y} = \{1, \ldots, K\}$, el objetivo de los algoritmos de clasificación es encontrar una aproximación que mapea $\mathcal{X}^D \rightarrow \mathcal{Y}$.

En el reconocimiento de patrones destacan el reconocimiento de imágenes o texto, áreas en las que se ha investigado mucho durante las últimas décadas (Aggarwal and Zhai, 2012; Alzubaidi et al., 2021; Smith et al., 2021). Sin embargo, los datos que manejamos de manera habitual en nuestro día a día y que más industrializados están, son los datos tabulares, es decir, conjuntos con una serie de variables que pueden influir en la etiqueta que se le da a una instancia de los datos. Existen diversos métodos que se pueden aplicar al reconocimiento de patrones, como por ejemplo los árboles de decisión. Los árboles de decisión para la clasificación son equivalentes a los árboles de decisión regresores. A diferencia de estos últimos, los espacios que se generan al segmentar los datos con reglas en el caso de los árboles de clasificación, engloban instancias con la misma clase del espacio discreto de etiquetas. También existen algoritmos más complejos como son las redes neuronales o los conjuntos de clasificadores (o *ensemble learning*). En estos últimos son en los que este proyecto se ha basado.

2.4. *Ensemble learning*

Uno de los paradigmas del aprendizaje automático más exitosos es el *ensemble learning* o algoritmos basados en conjuntos de clasificadores (Bentéjac et al., 2021; Fernández-Delgado et al., 2014). Este tipo de algoritmos combinan las decisiones de modelos más sencillos para alcanzar una predicción final. La forma de combinar las decisiones puede realizarse por votación, es decir, se escoge la etiqueta más común entre los modelos base combinados, para un ejemplo concreto, o realizando la media de las probabilidades que obtienen los algoritmos base para cada etiqueta del conjunto, seleccionando finalmente la etiqueta que mayor porcentaje obtiene. En regresión, la combinación de etiquetas suele realizarse mediante la media de las salidas de todos los modelos combinados en el conjunto. Existe una amplia evidencia que respalda que los conjuntos de clasificadores o de regresores tienden a superar a los algoritmos base combinados (Breiman, 1996b, 2001; Freund and Schapire, 1997).

Es posible que los clasificadores base de un conjunto no tengan una precisión elevada de manera individual. Sin embargo, la combinación de estos modelos subóptimos puede ayudar a la optimización de los problemas que se quieren resolver. Si se combina una serie de modelos con un comportamiento similar, se obtendrá un resultado parecido para todos los clasificadores base y, por lo tanto, la combinación de estos no aportará una mejora significativa. Para evitar esto, es necesario realizar transformaciones o bien en los datos con los que se entrena cada clasificador base o en los algoritmos combinados, o bien en el proceso secuencial de construcción de los clasificadores. En general

podemos hablar principalmente de dos métodos para la creación de clasificadores diversos: optimización y aleatorización. Ambas estrategias han obtenido resultados competitivos. En las técnicas de optimización destacan algoritmos como AdaBoost (Freund and Schapire, 1997), que modifica el peso de las instancias mal clasificadas durante el entrenamiento de los modelos base para focalizarse en los ejemplos más complicados de clasificar. También destaca Gradient Boosting y sus variantes XGBoost y CatBoost, que generan un modelo aditivo de forma iterativa minimizando una función de pérdida (Chen and Guestrin, 2016; Friedman, 2001; Prokhorenkova et al., 2018). En las técnicas de aleatorización podemos encontrarnos con varios métodos. Existen algoritmos que realizan modificaciones en los clasificadores que combinan y otros algoritmos aumentan la diversidad de los modelos base realizando modificaciones en los datos de entrenamiento. Algunos modelos que destacan son Bagging (Breiman, 1996b), Class-switching (Martínez-Muñoz and Suárez, 2005) o el algoritmo Random Forest (Breiman, 2001) que además realiza muestreos bootstrap para generar cada clasificador. Además, se ha demostrado que aumentando la diversidad de los clasificadores combinados realizando cierta aleatorización en las etiquetas de los datos de entrenamiento, se pueden obtener resultados de generalización competitivos (Breiman, 1996a, 2000; Martínez-Muñoz and Suárez, 2005).

La mejora en el error de generalización dada por la combinación de clasificadores se puede explicar con la descomposición sesgo – varianza.

$$Error = E_{sesgo} + E_{varianza} + \epsilon$$

Esta descomposición explica que el error de generalización esperado para un clasificador se puede separar en un término de varianza, en un término de sesgo y en un error irreducible ϵ dado por los datos. Si bien esta descomposición es clara para regresión, no lo es tanto para clasificación. Se ha definido esta descomposición de diversas formas, pero la más difundida define el término del sesgo como el error *persistente* del algoritmo, es decir, el error que se mantendría aunque se utilizaran infinitos clasificadores independientemente entrenados. Y el término de la varianza explica el error que genera la diversidad de un único modelo. La idea de los conjuntos de clasificadores es que se puede reducir el término de la varianza, y *por ende* el error esperado, combinando los errores obtenidos por cada algoritmo individual (Kong and Dietterich, 1995; Schapire et al., 1998). Es por esto que la aleatorización de los modelos base tiende a mejorar los resultados obtenidos (Breiman, 1996b, 2001; Martínez-Muñoz and Suárez, 2005). Además, algoritmos basados en optimización también pueden reducir el término de sesgo (Schapire et al., 1998).

2.4.1. AdaBoost

El *Boosting* es una técnica que se enfoca en mejorar los algoritmos débiles, entendiendo como algoritmos débiles aquellos que obtienen un error de generalización ligeramente mejor que un algoritmo de clasificación aleatorio. El objetivo de los métodos basados en boosting es mejorar los algoritmos

combinados en base a los resultados obtenidos en los clasificadores previamente construidos (Breiman, 1996a).

Uno de los primeros algoritmos basados en técnicas de boosting fue AdaBoost (Freund and Schapire, 1997). En este algoritmo, todas las instancias del conjunto de datos con el que se entrena el primer clasificador individual tienen el mismo peso. Se realizan tantas iteraciones como clasificadores se quieran construir. Para cada iteración se construye un clasificador con todos los datos del problema y se ponderan por el peso correspondiente (inicializados todos al mismo valor para el primer clasificador). Se calcula el error ϵ_k para cada clasificador, como la suma de todos los pesos de los ejemplos mal clasificados. Si el error ϵ_k obtenido es $\epsilon_k \geq 0{,}5$ o $\epsilon_k = 0$ el algoritmo termina y no se incluye este clasificador en el conjunto. Si el error obtenido no cumple estas condiciones, entonces se reasignan los pesos de las instancias en base a este error obtenido. Se ponderan los ejemplos mal clasificados multiplicando el peso que tenía antes la instancia por β_k, siendo $\beta_k = \frac{\epsilon_k}{(1-\epsilon_k)}$. Los ejemplos que estaban bien clasificados mantienen el peso anterior. Finalmente, la clasificación de las instancias se realiza mediante el voto ponderado con $\log \frac{1}{\beta_k}$, de forma que los clasificadores con menos error cuentan más en la decisión final.

2.4.2. Gradient Boosting

Gradient Boosting es otro algoritmo basado en optimización. A diferencia de AdaBoost, este modelo no entrena clasificadores, sino que entrena regresores. Estos regresores se entrenan de forma que vayan minimizando una función de pérdida. Para el caso de usar el error cuadrático medio como función de pérdida, el primer modelo es la media de las salidas a aprender y para cada iteración t entrena un nuevo modelo con los valores modificados como $y_i - H_{t-1}(x_i)$, donde H es el modelo combinado anterior. De esta forma, en cada iteración se entrena un nuevo regresor que intenta aprender los residuos del modelo anterior. La forma de clasificar las instancias del problema es utilizando una función sigmoidal, de forma que se suma el resultado de los regresores y al pasarlo por la sigmoidal obtenemos una probabilidad entre 0 y 1 de que la instancia pertenezca a la etiqueta 1.

2.4.3. Bagging

Bagging es un método de *ensemble learning* muy eficaz desarrollado por Breiman (1996b). Este algoritmo construye cada árbol de decisión con muestras aleatorias de los datos originales. Estas muestras, denominadas muestras *bootstrap*, se seleccionan realizando extracciones aleatorias de las instancias de los datos con repetición. Esto quiere decir que una fracción de las instancias seleccionadas van a estar repetidas en la muestra. Así, cada árbol de decisión es entrenado utilizando un conjunto de datos diferente, aumentando así la diversidad de estos y disminuyendo el error de varianza del modelo completo. Este método no sólo es eficiente por sí mismo, sino que también es muy sencillo

combinar las muestras bootstrap con otros conjuntos de clasificadores.

2.4.4. Random Forest

Random Forest es un algoritmo de clasificación basado en la combinación de árboles de decisión. Se ha demostrado que no produce sobreajuste con respecto al número de clasificadores que se combinan y que tiende a la convergencia en el error de generalización (Breiman, 2001). La forma de introducir diversidad en los árboles de decisión combinados por Random Forest es utilizando un subconjunto aleatorio del espacio de atributos del problema para cada nodo de cada árbol de decisión y escogiendo la regla de separación óptima de los datos de entre los atributos de este subconjunto. De esta forma, cada clasificador combinado es diferente al anterior. Además, Random Forest también utiliza muestras bootstrap para la construcción de los conjuntos de datos de entrenamiento de cada árbol de decisión, por lo que aumenta también la diversidad de cada clasificador combinado. Otro de los aspectos reseñables de este modelo es que la independencia de cada clasificador base para con el resto de árboles permite la optimización del algoritmo mediante la paralelización del entrenamiento de los modelos. Random Forest combina las etiquetas de salida de cada modelo base para escoger la etiqueta final de cada instancia mediante voto por mayoría (Breiman, 2001). El algoritmo de Random Forest ha demostrado obtener resultados formidables en gran variedad de problemas y es considerado uno de los algoritmos más eficaces en el aprendizaje supervisado (Fernández-Delgado et al., 2014).

2.4.5. Class Switching

Class Switching también combina árboles de decisión para optimizar el error de generalización para el problema que se intente resolver. A diferencia de Random Forest, Class Switching es un modelo que modifica el conjunto de datos, en vez de modificar los clasificadores combinados. Este algoritmo está basado en el método de aleatorización de etiquetas propuesto por Breiman (2000). La metodología que sigue este algoritmos es la de intercambiar las etiquetas de los datos de entrenamiento para cada árbol de decisión combinado, de forma que aumenta la diversidad de los modelos individuales para optimizar el error de varianza del conjunto completo. Este modelo ha demostrado obtener notables resultados. Sin embargo, es necesario combinar ≈ 1000 árboles de decisión para obtener bajos ratios de error (Martínez-Muñoz et al., 2008; Martínez-Muñoz and Suárez, 2005).

3

Descripción del método propuesto

El enfoque que sigue el algoritmo propuesto no es el tradicional del aprendizaje supervisado en el que se intenta predecir una etiqueta para cada instancia de un conjunto de datos, sino que el objetivo que persigue es identificar si los datos están etiquetados de forma correcta o incorrecta.

El algoritmo propuesto en este proyecto consiste en la combinación de clasificadores más sencillos –árboles de decisión– y el objetivo que persigue es el de aprender a distinguir si una instancia del conjunto de datos está correctamente etiquetada o no. Para ello, los clasificadores base que se combinan se entrenan sobre un problema distinto al original, con la intención de que aprendan a distinguir si una instancia está correctamente etiquetada o no, en lugar de asignarle una etiqueta de forma directa. La idea esencial del nuevo problema es que para cada subconjunto de datos con el que se entrena cada árbol de decisión del conjunto, se seleccionan una serie de instancias y se modifica su etiqueta. Esta modificación de la etiqueta se incluye en el conjunto de variables como una clase sugerida, y la nueva etiqueta a predecir es ahora '1' si la instancia no ha sido modificada (es decir, la clase sugerida coincide con la etiqueta original de la instancia) y '0', para aquellas instancias que sí han sido modificadas (la clase sugerida no coincide con la etiqueta original de la instancia). En la figura 3.1 se muestra este proceso de modificación del conjunto de datos de entrenamiento. En el primer paso del algoritmo se modifican las etiquetas de una serie de instancias y en el segundo paso del algoritmo se añaden estas nuevas clases modificadas como atributo del conjunto de datos. Finalmente, se asigna una nueva etiqueta para las instancias del nuevo conjunto de datos. Con estas etiquetas se diferencian las instancias con la clase sugerida correcta o incorrecta.

3.1. Entrenamiento

La entrada de cada modelo base del algoritmo propuesto es el vector de la instancia junto a una *clase sugerida* para cada instancia. Cada modelo se entrena con el objetivo de identificar si el par $\langle$instancia, *clase sugerida*$\rangle$ es una asignación correcta o no. Específicamente, para entrenar cada algoritmo base, se crea un nuevo conjunto de datos como $Q' = \{\mathbf{x}'^{(i)}, z^{(i)}\}_{i=1}^{N}$ donde cada instancia, $\mathbf{x}'$, del conjunto de entrenamiento se compone de la instancia original, $\mathbf{x}$, y una clase sugerida, y'. Esto

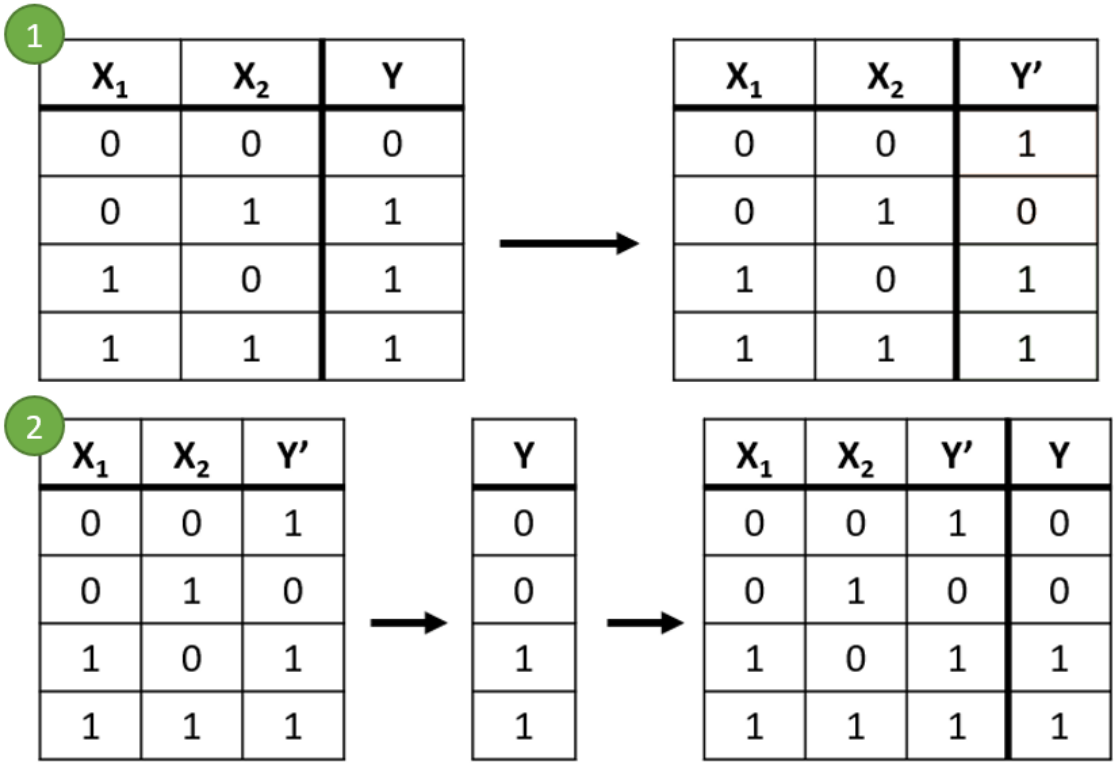


Figura 3.1: En el primer paso del entrenamiento se realiza el cambio de algunas clases del conjunto de datos. En el segundo paso, se añade la etiqueta modificada de las instancias como una nueva variable del conjunto de entrenamiento y se etiquetan los datos con '1' para los ejemplos con la clase sugerida no modificada y, '0', aquellas instancias con una clase sugerida distinta a la etiqueta original.

es, $\mathbf{x}' = \langle x_1, x_2, \ldots, x_D, y' \rangle$ con $y' \in \{1, \ldots, K\}$. Además, a la nueva etiqueta, z, se le asigna un valor $z = 0$ si la *clase sugerida* tiene un valor distinto al de la clase original de la instancia $\mathbf{x}$ (es decir, si $y' \neq y$), y se le asigna $z = 1$ si la *clase sugerida* es igual a la clase original de la instancia (es decir, si $y' = y$). En este trabajo se proponen dos métodos de conjuntos de clasificadores a partir de esta idea: el primero se basa en la aleatorización y el segundo además utiliza un algoritmo de vecinos próximos auxiliar para determinar la clase sugerida.

3.1.1. Método aleatorio

Cada uno de los modelos base del conjunto se entrena creando un conjunto de datos aleatorizado con las características previamente descritas de la siguiente forma: (1) se extrae una muestra bootstrap de tamaño N con reemplazamiento del dataset original Q. Este paso se utiliza para generar mayor diversidad en los clasificadores base del modelo, como en otros métodos de conjuntos como random forest y bagging; (2) se selecciona un porcentaje de instancias de la muestra bootstrap de forma aleatoria sin reemplazamiento; (3) se introduce, como un nuevo atributo, una clase sugerida aleatoria de $\mathcal{Y}$ diferente a la etiqueta original para las instancias seleccionadas en el paso anterior; (4) después se selecciona la clase original como la *clase sugerida* para las instancias que no han sido seleccionadas en el segundo paso del algoritmo; y (5) finalmente, se establece la etiqueta $z = 0$ para el primer subconjunto de las instancias (las instancias cuya *clase sugerida* ha sido seleccionada aleatoriamente) y la etiqueta $z = 1$ para el último subconjunto (las instancias cuya *clase sugerida* se corresponde con la clase original). Los pasos (1)-(5) se repiten para el entrenamiento de cada clasificador base del conjunto.

3.1.2. Método vecinos cercanos

En el método mejorado con vecinos cercanos, los dos primeros pasos del algoritmo son idénticos al método aleatorio. Es decir, (1) se extrae una muestra bootstrap de tamaño N con reemplazamiento del dataset original y (2) se selecciona un porcentaje de las instancias de la muestra. Sin embargo, los últimos pasos del algoritmo difieren del primer método. Tras seleccionar las instancias de la muestra bootstrap, se incluyen los siguientes pasos: (3) se seleccionan *k* instancias cercanas (mediante el método de vecinos cercanos o *nearest neighbors*) para cada instancia seleccionada en el paso anterior, y se escogen las etiquetas de estas instancias vecinas $\mathcal{Y}_k$ como posibles clases sugeridas con las que se modificarán las etiquetas de la muestra. Este conjunto de etiquetas extraídas de las instancias vecinas solo puede contener una o varias etiquetas **diferentes** a la de cada instancia objetivo (instancias seleccionadas para ser modificadas). Si todas las etiquetas de los vecinos cercanos de la instancia comparten la misma etiqueta que la instancia objetivo, el conjunto de etiquetas $\mathcal{Y}_k$ de esa instancia, es el conjunto completo de etiquetas del problema eliminando la etiqueta original de la instancia; (4) se introduce el nuevo atributo *clase sugerida* escogiendo una etiqueta aleatoria del conjunto de etiquetas

$\mathcal{Y}_k$ para cada instancia seleccionada. Finalmente, los dos últimos pasos del algoritmo son también idénticos al método aleatorio: (5) para las instancias no seleccionadas se mantiene la clase original como clase sugerida y (6) se establecen las nuevas etiquetas del subconjunto de forma que $z = 1$ para las instancias cuya clase sugerida se corresponde con la clase original y $z = 0$ para aquellas instancias seleccionadas, cuya clase sugerida difiere de la clase original del problema. Los pasos se repiten para el entrenamiento de cada clasificador base del conjunto.

3.2. Clasificación

El proceso de clasificación de nuevas intancias requiere una preparación previa de los datos, ya que los clasificadores base han sido entrenados sobre un problema distinto al original: el problema de aprender a distinguir si una etiqueta es correcta o no, para una instancia. Por lo tanto, dada una nueva instancia a clasificar $\mathbf{x}$, se crearán K nuevas instancias como $\mathbf{x} = \langle x_1, x_2, \ldots, x_D, k\rangle$ con $k = 1, \ldots, K$ siendo K el número de clases del problema original. Cada árbol de decisión del conjunto otorga una probabilidad $\mathbf{p}$ de que una instancia $\mathbf{x}$ esté correctamente clasificada, es decir, que la probabilidad de que la clase sugerida coincida con la clase real de la instancia. Esta probabilidad se mide como la fracción de ejemplos con la etiqueta $z = 1$ *(etiquetación correcta)* dada por una hoja de los árboles de decisión. En el modelo propuesto vamos a utilizar la media de las probabilidades dadas por todos los árboles del conjunto como la probabilidad final, por lo tanto, para cada instancia $\mathbf{x}_k$ existe un $\overline{p}_i = average\langle p_1, p_2, \ldots, p_J\rangle$ siendo J el número total del árboles de decisión combinados. Esto es la media de las probabilidades obtenidas con cada árbol de decisión del modelo. La media de estas probabilidades existe para cada clase sugerida posible (cada clase del problema), por lo que para cada instancia $\mathbf{x}$ existe un conjunto de probabilidades $p_k = \langle \overline{p}_1, \overline{p}_2, \ldots, K\rangle$, conjunto que representa las probabilidades medias obtenidas para cada clase del problema. La clase final escogida es, por lo tanto, $argmax_k\ p_k$ para la instancia $\mathbf{x}$, o lo que es lo mismo, la etiqueta escogida para cada instancia es aquella que haya obtenido la media de probabilidades más alta entre todos los clasificadores combinados.

3.3. Parámetros

El algoritmo propuesto tiene una serie de hiperparámetros que pueden ser establecidos. El primer hiperparámetro del modelo es el número de clasificadores base que van a ser entrenados. Por defecto, este parámetro está establecido en 100 clasificadores. El nombre de este hiperparámetro es ***n_estimators***. Otra variable que podemos modificar para el entrenamiento del modelo es la muestra ***bootstrap***. Este parámetro admite un número decimal y establece la fracción de instancias que se extraen (con repetición) del conjunto de datos de entrenamiento para crear cada subconjunto de

datos con el que se entrena cada modelo base. Al tratarse de extracciones con repetición, el número de instancias de cada subconjunto puede ser mayor que el conjunto de entrenamiento utilizando $bootstrap > 1.0$.

Existen tres hiperparámetros del modelo que se combinan para calcular el número de instancias que se van a modificar durante el entrenamiento de cada modelo base. Estos hiperparámetros se llaman: *method*, *pil* y *nearest_neighbours*. El hiperparámetro *pil* es un porcentaje y, en combinación con el hiperparámetro *method*, indica la fracción de instancias que se van a modificar en el subconjunto de datos de entrenamiento para cada árbol de decisión del conjunto. El hiperparámetro *method* indica el método de aleatorización. Se puede diferenciar entre cinco técnicas de aleatorización distintas, tres englobadas en el método aleatorio y dos, en el método de vecinos cercanos.

3.3.1. Parámetros para el método aleatorio

El método aleatorio de entrenamiento de los modelos engloba tres técnicas distintas para la modificación de las instancias de los conjuntos de entrenamiento:

- ***regular***: Se modifica la fracción de instancias indicado por el porcentaje del parámetro *pil*, cambiando las clases de estas instancias por las de otras instancias del subconjunto (sin incluir la clase original). Las clases son escogidas de forma aleatoria. En el algoritmo 3.1 se describe el pseudocódigo del método *regular*. Este método recibe tres parámetros: los atributos X, las etiquetas originales de los datos Y y el parámetro *pil* representado en el algoritmo como PIL. El algoritmo describe lo siguiente: se generan tantos clasificadores como se indique en $n\_estimators$, se generan las modificaciones de las etiquetas de la forma descrita previamente y se añaden como nuevo atributo al conjunto X. Si la nueva clase sugerida es igual que la etiqueta original Y, se le asigna '1' como nueva clase y, sino, '0'. Finalmente, se entrena cada clasificador base con los datos modificados y se añade al conjunto.

```
1  Function entrenamiento_regular( X, Y, PIL )
       input: Conjunto de datos X de tamaño tam_conjunto × num_atributos
       input: Conjunto de etiquetas Y de tamaño tam_conjunto
       input: Probabilidad de modificación aleatoria de los datos PIL
2      for i ← 0 to #clasificadores do
3          clasificador ← new Clasificador();
4          clase_sugerida ← clase_aleatoria( Y, PIL );
5          x_modificado ← añadir_atributo( X, clase_sugerida );
6          y_modificado ← Y == clase_sugerida;
7          clasificador.entrenamiento( x_modificado, y_modificado );
8          conjunto_clasificadores [i] ← clasificador;
9      end
```

Algoritmo 3.1: Pseudocódigo del algoritmo de entrenamiento del modelo para el método regular.

- ***randomized***: Para cada instancia del subconjunto de datos se extrae una etiqueta de entre

todas las etiquetas de las instancias del subconjunto, sin repetición. Esto sería equivalente a que se modifique el orden de todas las clases de las instancias del subconjunto, pero no de las instancias en sí: una permutación aleatoria de las etiquetas del subconjunto de datos. En el algoritmo 3.2 se describe el pseudocódigo del método *randomized*. A diferencia del método regular, no recibe el parámetro *pil*, sólo el conjunto de atributos de los datos y sus respectivas etiquetas. Difiere del método regular en que la clase sugerida se calcula realizando la función *shuffle* (o "barajar") sobre el conjunto de etiquetas de los datos.

```
1  Function entrenamiento_randomized( X, Y )
       input: Conjunto de datos X de tamaño tam_conjunto × num_atributos
       input: Conjunto de etiquetas Y de tamaño tam_conjunto
2      for i ← 0 to #clasificadores do
3          clasificador ← new Clasificador();
4          clase_sugerida ← shuffle( Y );
5          x_modificado ← añadir_atributo( X, clase_sugerida );
6          y_modificado ← Y == clase_sugerida;
7          clasificador.entrenamiento( x_modificado, y_modificado );
8          conjunto_clasificadores [i] ← clasificador;
9      end
```

Algoritmo 3.2: Pseudocódigo del algoritmo de entrenamiento del modelo para el método *randomized*.

- ***distributed***: El objetivo de este método es mantener la proporción de las clases tras completar la aleatorización del subconjunto de entrenamiento. Para ello, a cada instancia de los datos se le asigna una probabilidad $p(j|k); j \neq k$, de que si su clase es k sea modificada por otra clase j. Este método proviene de (Breiman, 2000) y se aplican las siguientes fórmulas para obtener $p(j|k)$:

$$p(j|k) = w \cdot c(j), j \neq k$$

$p(j|k)$ indica la probabilidad de que la clase k sea modificada a una clase $j, k \neq j$, $c(j)$ indica el porcentaje de instancias etiquetadas como j del problema y w se calcula con el siguiente método:

$$w = \frac{pil}{(1 - \Sigma_j c(j)^2)}$$

Para extraer w, sabemos que la proporción de instancias con la clase k que va a ser modificada es $w \cdot (1 - c(k))$. La proporción total de instancias que van a ser modificadas es la suma de $w \cdot (1 - c(k)) \cdot c(k)$. *pil* es el parámetro del modelo que indica la fracción de instancias que van a ser modificadas. Si igualamos $pil = w \cdot (1 - c(k)) \cdot c(k)$ y despejamos w, tenemos $w = \frac{pil}{(1-\Sigma_j c(j)^2)}$. Una vez calculado $p(j|k)$, se selecciona un número aleatorio entre 0 y 1, si el número es superior a $p(j|k)$ se modifica la etiqueta de la instancia. En el algoritmo 3.3 se describe el pseudocódigo del método *distributed*.

```
Function entrenamiento_distributed( X, Y, PIL )
    input: Conjunto de datos X de tamaño tam_conjunto × num_atributos
    input: Conjunto de etiquetas Y de tamaño tam_conjunto
    input: Probabilidad de modificación aleatoria de los datos PIL
    for i ← 0 to #clasificadores do
        clasificador ← new Clasificador();
        for class_k in unique( Y ) do
            class_j ← (1 − class_k);
            w ← calcular_w( Y, class_j, PIL );
            probabilidad_cambio ← w · distribucion( Y, class_j );
            clase_sugerida ← clase_aleatoria( Y, probabilidad_cambio );
            x_k_modificado ← añadir_atributo( X, clase_sugerida );
            y_k_modificado ← Y == clase_sugerida;
            x_modificado [k] ← x_k_modificado;
            y_modificado [k] ← y_k_modificado;
        end
        clasificador.entrenamiento( x_modificado, y_modificado );
        conjunto_clasificadores [i] ← clasificador;
    end
```

Algoritmo 3.3: Pseudocódigo del algoritmo de entrenamiento del modelo para el método *distributed*.

3.3.2. Parámetros para el método de vecinos cercanos

El método de entrenamiento que utiliza la aproximación de vecinos cercanos, engloba dos técnicas para la modificación de las muestras de cada algoritmo base combinado:

- ***nearest_neighbors***: Con este método primero se selecciona una fracción de instancias a modificar indicado por el parámetro *pil*. A diferencia del resto de métodos, se seleccionan k datos vecinos (mediante el algoritmo de *nearest_neighbors* implementado en scikit-learn (Pedregosa et al., 2011)) para cada instancia que se va a modificar. Para cada instancia a modificar se escoge una etiqueta de forma aleatoria de alguno de los datos vecinos. Si todos los vecinos escogidos tienen la misma etiqueta que la instancia a modificar, se selecciona de forma aleatoria de entre todas las clases posibles una nueva etiqueta distinta a la clase original de la instancia. El algoritmo 3.4 describe el comportamiento en pseudocódigo del método propuesto utilizando vecinos cercanos para la modificación de los conjuntos de entrenamiento. En el pseudocódigo del algoritmo se puede ver que para cada instancia X_j del subconjunto seleccionado a modificar, se extraen las etiquetas de sus k vecinos cercanos y se escoge de entre estas aleatoriamente una clase sugerida para el ejemplo. Si la única etiqueta que comparten todas las instancias vecinas es igual que la etiqueta original, se escoge de entre el conjunto completo de etiquetas de forma aleatoria.

```
1  Function entrenamiento_nearest_neighbors( X, Y, PIL, k )
       input: Conjunto de datos X de tamaño tam_conjunto × num_atributos
       input: Conjunto de etiquetas Y de tamaño tam_conjunto
       input: Probabilidad de modificación aleatoria de los datos PIL
       input: Número de vecinos sobre los que ejecutar el algoritmo knn k
2      for i ← 0 to #clasificadores do
3          clasificador ← new Clasificador();
4          for j ← 0 to tam_conjunto do
5              vecinos ← clases_vecinos_cercanos( X_j, k );
6              if Y_j == unique( vecinos ) then
7                  clase_sugerida_j ← clase_aleatoria( Y, PIL );
8              end
9              else
10                 clase_sugerida_j ← clase_aleatoria( vecinos, PIL );
11             end
12             x_modificado [j] ← añadir_atributo( X, clase_sugerida_j );
13             y_modificado [j] ← Y_j == clase_sugerida_j;
14         end
15         clasificador.entrenamiento( x_modificado, y_modificado );
16         conjunto_clasificadores [i] ← clasificador;
17     end
```

Algoritmo 3.4: Pseudocódigo del algoritmo de entrenamiento del modelo para el método *nearest_neighbors*.

- ***no_pil***: El comportamiento de este método de aleatorización es idéntico al, previamente descrito *nearest_neighbors* excepto que no se utiliza *pil* para la selección de las instancias a modificar. En este caso, se seleccionan los k vecinos cercanos de todas las instancias y se escoge una etiqueta aleatoria de entre las clases de estos vecinos para cada instancia. Si todas las instancias vecinas tienen la misma etiqueta, no se modifica la instancia. De esta forma se orienta el problema hacia aquellas instancias más próximas a la frontera de decisión, y por ende, aquellas instancias más complicadas de clasificar. En el algoritmo 3.5 se describe el pseudocódigo de este método. Como se puede observar, la única diferencia con el método de nearest neighbors es que sólo se modifican las etiquetas de aquellos ejemplos cuyos vecinos tengan alguna etiqueta distinta a la original del ejemplo.

```
Function entrenamiento_nearest_neighbors( X, Y, k )
    input: Conjunto de datos X de tamaño tam_conjunto × num_atributos
    input: Conjunto de etiquetas Y de tamaño tam_conjunto
    input: Número de vecinos sobre los que ejecutar el algoritmo knn k
    for i ← 0 to #clasificadores do
        clasificador ← new Clasificador();
        for j ← 0 to tam_conjunto do
            vecinos ← clases_vecinos_cercanos( X_j, k );
            if Y_j in vecinos then
                clase_sugerida_j ← clase_aleatoria( vecinos );
            end
            else
                clase_sugerida_j ← Y_j;
            end
            x_modificado [j] ← añadir_atributo( X, clase_sugerida_j );
            y_modificado [j] ← Y_j == clase_sugerida_j;
        end
        clasificador.entrenamiento( x_modificado, y_modificado );
        conjunto_clasificadores [i] ← clasificador;
    end
```

Algoritmo 3.5: Pseudocódigo del algoritmo de entrenamiento del modelo para el método *no_pil*.

3.4. Diseño e implementación

El modelo implementado asigna la etiqueta 0 si una instancia está mal clasificada o 1, si la instancia está correctamente clasificada. Como se trata de un algoritmo de clasificación, lo ideal es que la interacción que nos ofrezca el modelo no sólo sean las etiquetas 0 o 1, sino que nos ofrezca la etiqueta correcta de las instancias. Por esta razón, el modelo se ha implementado en `python` en forma de librería para que su utilización sea lo más sencilla posible y la implementación del algoritmo se ha encapsulado siguiendo los estándares de la librería `scikit-learn` (Pedregosa et al., 2011) para que la salida del método sean las etiquetas predichas de las instancias del problema. Esto es, el modelo implementa un método constructor en el que se establecen los parámetros con los que se va a realizar el entrenamiento del modelo, un método `fit` que se encarga de entrenar el conjunto de clasificadores completo, un método `predict_proba` que devuelve la probabilidad para cada etiqueta del problema de que una instancia sea clasificada con dicha etiqueta y un método `predict`, que utiliza los valores devueltos por el método `predict_proba` y devuelve la clase con mayor probabilidad. Además de estos métodos, la clase implementada también incorpora una serie de métodos privados que utilizan los parámetros del modelo para realizar la aleatorización pertinente durante el entrenamiento de los árboles de decisión.

Además, se ha incorporado un parámetro en el modelo (*n_jobs*) que permite la paralelización del entrenamiento del modelo. Esto hace que el coste computacional que conlleva el entrenamiento se vea reducido de manera drástica.

3.5. Regresión

A partir del algoritmo propuesto de clasificación, se ha intentado elaborar un algoritmo de regresión basado en los mismos conceptos: conjuntos de clasificadores, aleatorización de etiquetas de salida y sugerencia de etiquetado en la clasificación (en este caso en la regresión).

El objetivo de este algoritmo era combinar una serie de regresores basados en árboles de decisión, modificar las etiquetas de los datos de entrenamiento de forma aleatoria y añadirlos como una nueva variable del conjunto de datos con el que entrenar cada regresor. A diferencia del algoritmo de clasificación, con el entrenamiento de la regresión no es factible indicar si una clase sugerida es "correcta" o "incorrecta", sino que tenemos que indicar que el valor es cercano a la etiqueta continua real. La forma más sencilla de implementar esto era utilizar el valor de la diferencia entre la etiqueta original y la etiqueta sugerida. Así, si el resultado es cercano a 0 indica que el valor sugerido es cercano al valor original.

El entrenamiento de este modelo sigue una serie de pasos similares a los que se utilizan para el algoritmo original de clasificación, en este caso para cada regresor del conjunto: (1) se extrae una muestra bootstrap del conjunto de datos original de tamaño N y con reemplazamiento; (2) se selecciona un porcentaje *pil* de instancias cuyos valores de etiqueta van a ser modificados y posteriomente, añadidos como valor sugerido en una nueva variable del conjunto; (3) se recogen las etiquetas de los k vecinos cercanos para cada instancia que va a ser modificada y se escoge de entre estas etiquetas el nuevo valor sugerido de la instancia –este paso varía en comparación con REDOL de clasificación aleatorio porque escoger los nuevos valores sugeridos de entre el resto de valores originales para un problema de regresión puede conllevar una gran pérdida de eficiencia– si el nuevo valor sugerido es igual que el original, se escoge un valor aleatorio de entre el resto de valores del conjunto de datos; (4) se establece como nueva etiqueta de salida el valor de la diferencia entre el valor sugerido y el valor original de la instancia. De esta forma, las instancias etiquetadas con 0 son las instancias que no han sido modificadas, mientras que las instancias con valores de etiqueta $Z \neq 0$ serán aquellas que han sido modificadas.

Para aplicar el modelo a nuevas instancias es necesario introducir una nueva variable que sería la clase sugerida. En el modelo de clasificación esto se realiza añadiendo una clase sugerida por cada etiqueta del conjunto de entrenamiento. Sin embargo, al igual que en el entrenamiento, este proceso resultaría excesivamente costoso y se ha aplicado una estrategia alternativa en la que se itera sobre la regresión tantas veces como se indique por un parámetro del modelo, *num_repeticiones*. Sobre estas repeticiones se añade un valor aleatorio como valor sugerido de entre los posibles que se han dado en entrenamiento. Como el modelo regresor devuelve valores cercanos a 0 para aquellos valores correctos, lo que se hace es sumar el valor sugerido al valor devuelto.

Este método se ha evaluado sobre el conjunto de datos *housing*, del repositorio de UCI de *datasets*

(Dua and Graff, 2017) con la métrica RMSE. El método se ha comparado con el RMSE obtenido con un único regresor basado en árboles de decisión (iguales que los que combina el modelo) como *baseline*. Sin embargo, los resultados obtenidos del modelo no eran mejores que los expuestos en el *baseline* y se ha considerado el experimento no exitoso. Por ello, no se incluyen los experimentos en los resultados de esta memoria.

4

EXPERIMENTOS

El rendimiento del algoritmo propuesto ha sido analizado a través de una serie de experimentos. Estos experimentos han sido diseñados para entender los diferentes aspectos del algoritmo en profundidad. El primero de los experimentos analiza cómo evoluciona el modelo cuando se utiliza el método *regular* y se varía el parámetro *pil* de aleatorización, además se estudia la evolución del error debido al incremento del número de clasificadores. Este primer experimento también analiza la diversidad de los clasificadores individuales que componen el conjunto completo con respecto a la diversidad que se obtiene en los algoritmos base de Random Forest. En el segundo experimento, el método propuesto se compara con modelos basados en conjuntos de clasificadores que definen el estado del arte: Random Forest, Bagging y Gradient Boosting. Aunque las redes neuronales tienden a obtener notables resultados en tareas de reconocimiento de texto o de imágenes, tienen un bajo rendimiento en datos tabulares (Zhang et al., 2017). Por lo tanto, hemos considerado no introducirlos en el análisis comparativo de los modelos. En el tercer experimento realizado, se muestra la evolución de la frontera de decisión generada con respecto al número de clasificadores escogido. Finalmente, en el cuarto y último experimento realizado, se ha realizado una búsqueda de hiperparámetros del algoritmo y de Random Forest para una serie de conjuntos de datos. Se ha escogido el algoritmo de Random Forest con el objetivo de comparar un algoritmo similar al propuesto en este proyecto y que, a su vez, obtenga resultados competitivos.

4.1. Conjuntos de datos y configuración

Para tener una visión clara del rendimiento del algoritmo, todos los experimentos se han realizado sobre 15 problemas diferentes. Cuatro de los problemas analizados, *twonorm*, *threenorm*, *ringnorm* y *waveform*, se tratan de conjuntos de datos o *datasets* sintéticos propuestos por Breiman (1996a). El resto de problemas analizados provienen del repositorio de *datasets* UCI (Dua and Graff, 2017). En la tabla 4.1 se da una breve descripción de estos conjuntos de datos incluyendo el número de instancias, atributos, clases y la distribución de las clases.

Con el objetivo de conseguir resultados estables y precisos, para todos los experimentos, excep-

Tabla 4.1: Characteristics of the analyzed datasets

Dataset	Instancias	Atributos	Clases	Distribución de las clases
Wine	178	13	3	59/71/48
New-thyroid	215	5	3	150/35/30
Heart	270	13	2	150/120
Ionosphere	351	34	2	126/225
Australian	690	14	2	56/44
Wdbc	699	9	2	357/212
Diabetes	768	8	2	500/268
Tic-tac-toe	958	9	2	332/626
German	1000	20	2	700/300
Segment	2310	19	7	*eq. distrib*
Magic04	19020	10	2	12332/6688
Twonorm	5000	20	2	2500/2500
Threenorm	5000	20	2	2500/2500
Ringnorm	5000	20	2	2479/2521
Waveform	5000	20	3	1657/1647/1696

tuando la comparación de algoritmos con parámetros optimizados, se han realizado 100 iteraciones para cada conjunto de datos. Para el experimento de optimización de hiperparámetros se han realizado 10 iteraciones sobre todos los conjuntos de datos debido al elevado coste computacional del experimento. La media de la exactitud de las iteraciones realizadas conforma el resultado final de cada experimento para cada conjunto de datos. Cada iteración se ha llevado a cabo utilizando subconjuntos aleatorios de los datos. Los subconjuntos se dividen en 2/3 de los datos para el entrenamiento de los modelos y 1/3 de los datos para el conjunto de validación. Todos los subconjuntos divididos se han obtenido llevando a cabo una extracción estratificada de los datos con el propósito de mantener la proporción de etiquetado tanto en los conjuntos de entrenamiento como en los conjuntos de validación.

4.2. Análisis *pil* y método *regular*

El objetivo de este experimento es el de observar y entender cómo la variación de los valores del hiperparámetro *pil* afecta al rendimiento del algoritmo propuesto.

Este experimento se ejecuta realizando 100 iteraciones sobre cada conjunto de datos. Para cada iteración, el experimento sigue los siguientes pasos:

- El parámetro *pil* es asignado con valores desde $0{,}02$ hasta $0{,}98$ en pasos de $0{,}01$. El parámetro *method* se establece a *regular*. Por lo tanto, 97 conjuntos de clasificadores son entrenados, uno para cada valor *pil*.

- Para cada porcentaje *pil* se combinan 100 árboles de decisión en el conjunto de clasificadores, que se entrena sobre el subconjunto de datos de entrenamiento.
- Conjuntos de clasificadores de Random Forest (Breiman, 2001), Gradient Boosting (Friedman, 2001) y Bagging (Breiman, 1996b) compuestos por 100 clasificadores base, son también entrenados con el subconjunto de entrenamiento. Además, un único árbol de decisión se entrena como algoritmo base con el que comparar. Para esta parte, se utilizan la implementaciones de `scikit-learn` (Pedregosa et al., 2011). Estos algoritmos se entrenan utilizando los parámetros por defecto de *sklearn*.
- Todos los modelos entrenados se evalúan sobre el subconjunto de validación. El algoritmo propuesto se analiza de forma secuenciada para obtener su rendimiento para la combinación de 1 a 100 clasificadores base. El resto de conjuntos entrenados se analizan únicamente para la combinación de 100 clasificadores base.

Los resultados se muestran en dos gráficas distintas para cada conjunto de datos, como se observa en la fig. 4.1 para tres *datasets* representativos. Las gráficas de la izquierda muestran cómo varía el error con respecto al valor del parámetro de aleatorización (*pil*), que varía desde 2 % a 98 %. En estas gráficas se muestra el modelo utilizando un número distinto de clasificadores base. Específicamente, el conjunto compuesto por 1, 5, 10, 50 y 100 árboles. Además, también se muestran en la gráfica los errores de generalización medios para los conjuntos completos Random Forest (RF), Gradient Boosting (gB.) y Bagging (Bagg.) como líneas horizontales para tener una referencia de estos algoritmos. Las gráficas de la segunda columna de la fig. 4.1 representan la variación del error con respecto al número de árboles combinados para una serie de valores del parámetro *pil*. Para evitar un gráfico sobrecargado no se muestran todos los valores de *pil* probados. Exclusivamente se muestran los valores representativos de: 2 %, 5 %, 10 %, 50 % y 75 %. La figura 4.1 muestra los resultados para *Magic04* (primera fila), *Ionosphere* (segunda fila) y *Threenorm* (tercera fila). Los gráficos mostrados son representativos para todos los conjuntos de datos, que cualitativamente presentan resultados muy similares.

Como se puede ver en la columna de la izquierda de la figura 4.1, la tendencia general es que el error de generalización muestre un comportamiento simétrico alrededor de *pil*=50 %. Para valores bajos (y altos) del porcentaje de aleatorización, la tasa de error es bastante alta. A medida que este porcentaje se acerca a 50 %, la tasa de error disminuye rápidamente. Para conjuntos compuestos por 100 árboles de decisión, el error alcanza un valor mínimo estable desde un valor aproximado de 10–20 % a 80–90 %, dependiendo del conjunto de datos. Las diferencias en el rango y extensión de esta región son pequeñas entre los *datasets*. Para un único árbol de decisión, la tasa de error muestra un claro mínimo a *pil*$\approx$50 %. Una observación interesante es que, aunque los árboles individuales obtengan un rendimiento bajo, la combinación de estos en un conjunto consigue obtener resultados competitivos. Por ejemplo, para *pil*=80 %, un único árbol para el conjunto *Threenorm*, consigue $\approx$0.34 de error y combinando 100 árboles equivalentes a este, el error disminuye hasta $\approx$0.13. Esto indica que el procedimiento propuesto es capaz de generar árboles muy diversos que compensan sus limitaciones

individuales de forma eficiente. La diversidad del conjunto propuesto en relación al de Random Forest se puede observar en la figura 4.2. En esta figura se muestran diagramas Error-Kappa (Dietterich, 2000). Estos diagramas presentan para cada par de clasificadores base del conjunto, su error medio (eje vertical) y su estadística kappa (eje horizontal). Cuánto más bajos son los valores kappa, mayor es la diversidad para cada par de clasificadores. Como se puede ver en la fig. 4.2, el conjunto de clasificadores propuesto produce clasificadores base que son más débiles en cuando a error medio, pero más diversos que los generados por Random Forest.

Otra observación interesante para los conjuntos con 100 árboles de decisión y para algunos de los *datasets* probados (aprox. 1/3 de los conjuntos), es que la tasa de error mínimo se obtiene para los porcentajes simétricos $\approx$ 10-30 % y $\approx$ 70-90 %. Este efecto se puede observar en la fig. 4.1 para el experimento *Ionosphere* (tercera fila).

Como se observa en la columna derecha de la figura 4.1, la evolución del error de generalización decrece gradualmente a medida que el número de los modelos base combinados aumenta. Dependiendo del porcentaje de aleatorización, la convergencia de los conjuntos es más lenta o más rápida. Para los valores cercanos a 50 % la convergencia es más rápida. En los gráficos, la mayoría de los conjuntos convergen, excepto aquellos con valores bajos de *pil*.

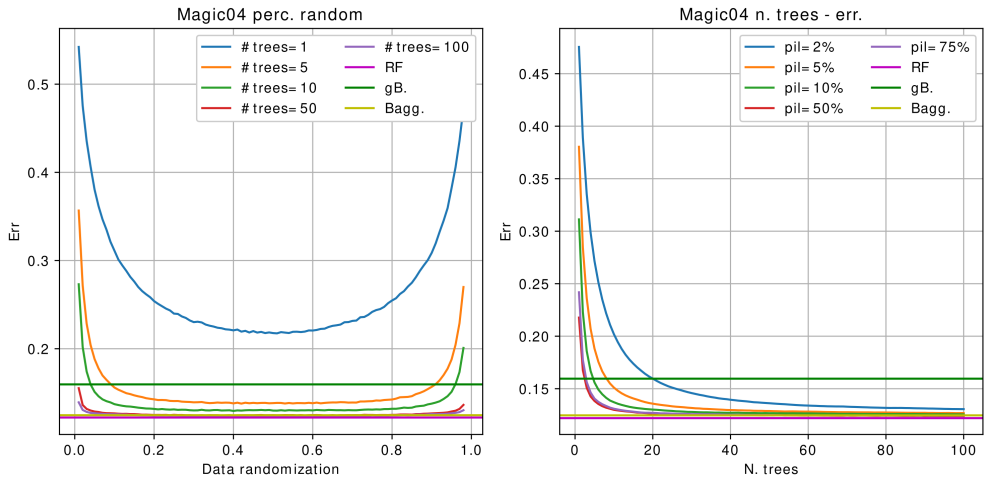
El coste computacional de este experimento es muy elevado. Esta es la razón por la que no se ha completado este mismo experimento con otros valores para el parámetro *method*. Sin embargo, el objetivo del experimento es observar el comportamiento del error al modificar el número de árboles combinados y el valor de *pil*, por lo que se puede concluir el éxito del experimento realizado.

4.3. Comparación de modelos

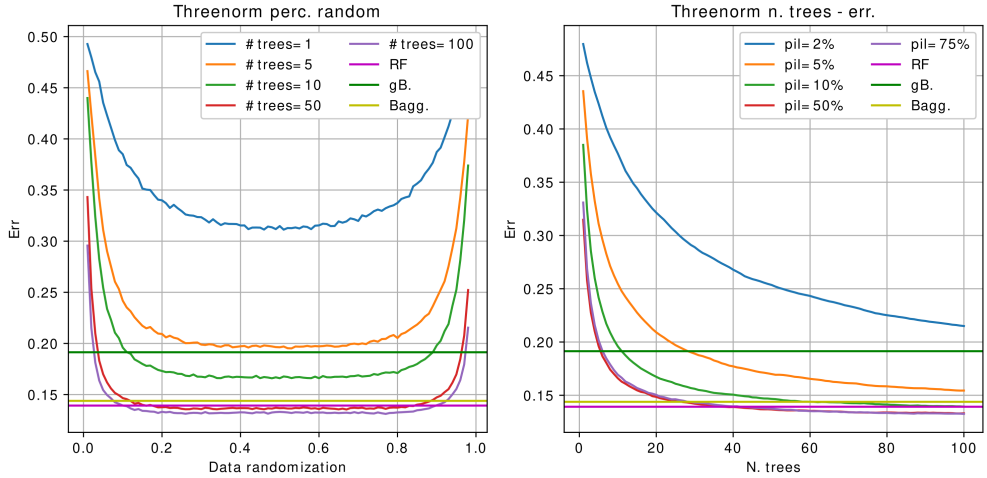
El propósito de este experimento es comparar el rendimiento del algoritmo propuesto con métodos que definen el estado del arte como Random Forest, Gradient Boosting y Bagging, y también, como referencia, con un único árbol de decisión. Como ya se ha descrito antes, todos los clasificadores son entrenados usando las mismas instancias de entrenamiento con el objetivo de conseguir resultados comparables. En estos experimentos, siguiendo las observaciones del anterior experimento, hemos usado un valor *pil*=75 % para todos los conjuntos de datos.

En la tabla 4.2 se pueden ver los resultados de la media de generalización para las 100 iteraciones realizadas sobre todos los conjuntos de datos. Nuestro método está etiquetado como REDOL (en inglés *Randomized Ensemble based on Distinguishing Output Labellings*). La desviación estándar se presenta después del símbolo $\pm$. Para cada conjunto de datos, el mejor resultado se resalta en negrita. El segundo mejor resultado está subrayado.

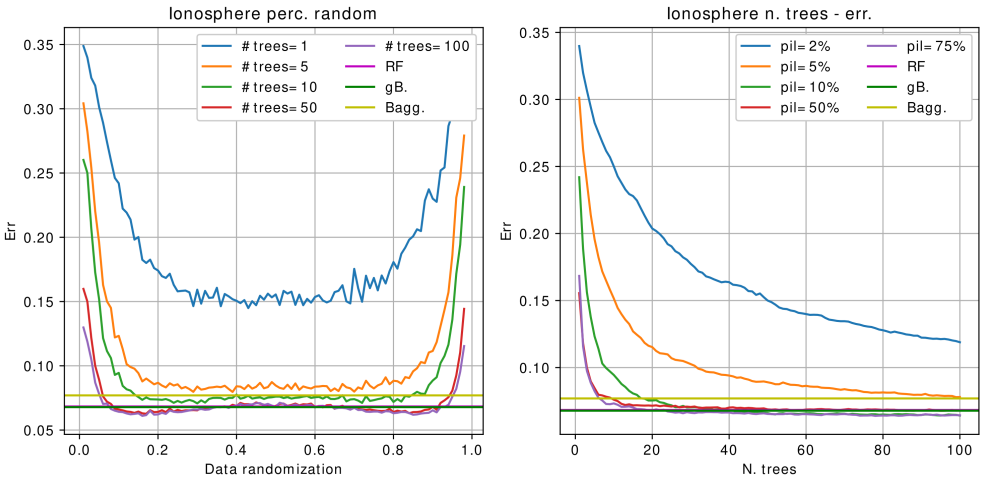
Como se puede ver en la tabla 4.2, las diferencias en el error entre los algoritmos estudiados es,



(a) Magic04

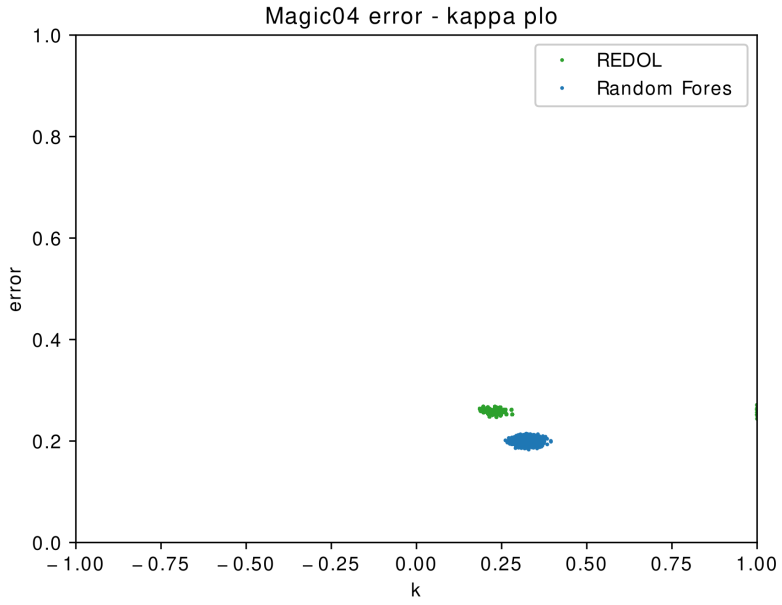


(b) Threenorm

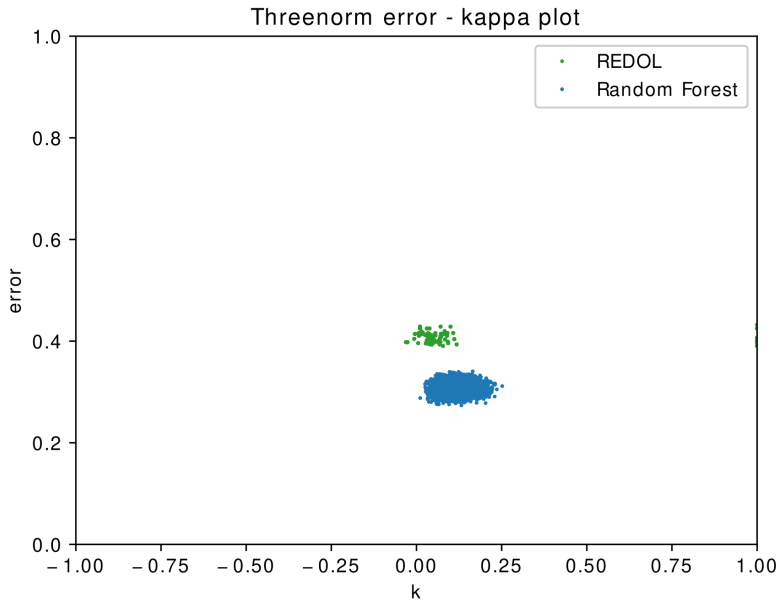


(c) Ionosphere

Figura 4.1: Análisis de los resultados obtenidos del hiperparámetro *pil* sobre el método *regular* sobre los conjuntos de datos *Magic04*, *Threenorm* e *Ionosphere*.



(a) Magic04



(b) Threenorm

Figura 4.2: Análisis de la diversidad que presentan los clasificadores base tanto del algoritmo propuesto como de Random Forest, para los conjuntos de datos *Magic04* y *Threenorm*.

Tabla 4.2: Comparación de los modelos. Se muestra la media del error de generalización junto a ± la desviación estándar. En **negrita** se resaltan los mejores resultados para cada conjunto de datos y subrayados se muestran los segundos mejores resultados.

Dataset	Dec. tree	REDOL	R. Forest	G. Boosting	Bagging
Wine	9.47±4.25	2.83±2.10	**2.13±1.74**	9.30±4.19	4.35±3.27
New-thyroid	7.53±2.81	7.05±2.73	**4.71±2.30**	7.04±2.77	5.88±3.01
Heart	25.94±4.23	17.51±3.20	**17.31±3.19**	20.54±2.89	19.35±3.39
Ionosphere	12.14±2.82	**6.43±2.04**	6.83±2.06	6.78±1.87	7.69±2.29
Australian	18.33±2.48	**12.88±1.92**	12.97±1.91	13.87±1.86	13.14±2.11
Wdbc	7.72±1.78	4.62±1.56	4.30±1.39	**3.59±1.77**	4.65±1.47
Diabetes	30.16±2.84	**23.20±2.19**	23.73±2.24	26.67±2.11	23.90±2.25
Tic-tac-toe	12.69±2.68	8.81±1.61	6.76±1.56	**0.77±0.66**	5.35±1.62
German	31.33±2.29	**23.18±1.64**	24.09±1.51	25.93±2.03	24.23±1.86
Segment	4.35±0.81	3.30±0.67	2.47±0.59	**2.16±0.59**	3.04±0.66
Magic04	18.52±0.51	12.43±0.36	**12.19±0.36**	15.95±0.41	12.46±0.36
Twonorm	16.81±0.92	**2.99±0.36**	3.13±0.37	4.13±0.45	3.61±0.51
Threenorm	29.12±1.11	**13.25±0.68**	13.92±0.72	19.13±0.66	14.38±1.62
Ringnorm	13.00±0.92	**3.39±0.50**	4.31±0.61	5.48±0.52	4.99±0.66
Waveform	24.91±1.04	**14.70±0.79**	14.88±0.69	16.40±0.79	16.06±0.81

en general, pequeña. Los resultados de generalización para un único árbol de decisión son, como esperábamos, los peores para todos los *datasets*. Sin embargo, estos resultados son mejores que los obtenidos para un único árbol de decisión del método propuesto, como se puede ver en la figura 4.1 (columna izquierda) para los conjuntos de datos *Magic04*, *Threenorm* y *Ionosphere*. El algoritmo propuesto obtiene los mejores resultados en ocho conjuntos de datos: *Ionosphere*, *Australian*, *Diabetes*, *German* y en todos los conjuntos de datos sintéticos. Además, el algoritmo propuesto obtiene los segundos mejores resultados en tres conjuntos de datos adicionales. Random Forest es el segundo mejor algoritmo en cuanto al número de mejores resultados obtenidos, consiguiendo los resultados más favorables en cuatro conjuntos de datos. Sin embargo, Random Forest, es el segundo mejor algoritmo en nueve conjuntos de datos. Esto demuestra que Random Forest es uno de los mejores métodos de clasificación, (Fernández-Delgado et al., 2014; Zhang et al., 2017). Aunque no obtenga el mejor error, es generalmente cercano al mejor. Los conjuntos basados en Boosting consiguen los mejores resultados en tres *datasets*. Finalmente, Bagging no alcanza la tasa de error más baja en ninguno de los conjuntos de datos analizados.

4.4. Frontera de decisión

Para analizar cómo evolucionan las fronteras de decisión del modelo propuesto a medida que aumenta el número de clasificadores combinados, se ha llevado a cabo este experimento con el conjunto de datos *Waveform*. Para ser capaces de hacer una gráfica de los resultados, el conjunto de datos se transforma a 2 dimensiones utilizando el algoritmo PCA. Para este experimento, el modelo es entrenado con 1, 10, 100 y 1000 clasificadores utilizando *pil*=75 % y método *regular*. Los resultados se presentan en la figura 4.3 en cuatro gráficas distintas para 1, 10, 100 y 1000 clasificadores base, de izquierda a derecha y desde arriba hacia abajo, respectivamente. En las gráficas, cada una de las tres clases del problema se muestra de un color diferente. Además, los ejemplos de entrenamiento también se muestran en las gráficas para tener una referencia.

En la fig.4.3 lo que vemos es cómo la frontera de decisión se suaviza a medida que el número de clasificadores combinados para el conjunto aumenta. Debido a las características de los árboles de decisión, la frontera de decisión forma líneas paralelas a los ejes, lo cuál puede verse de forma más clara cuándo sólo se combina un único árbol de decisión (ver la gráfica arriba–izquierda de la fig. 4.3). Sin embargo, a medida que el número de clasificadores aumenta, la frontera de decisión se suaviza. Aunque existen diferencias entre las cuatro gráficas, se observan menos diferencias entre la gráfica de 100 clasificadores y el conjunto de 1000 clasificadores.

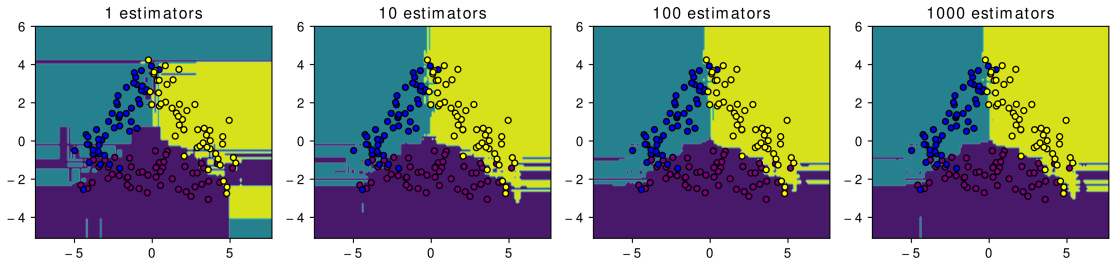


Figura 4.3: Frontera de decisión generada con 1, 10, 100 y 1000 árboles de decisión para el conjunto de datos *Waveform*.

4.5. Búsqueda y comparación de hiperparámetros

Finalmente, para realizar una comparación exhaustiva del modelo propuesto se ha llevado a cabo un último experimento. El objetivo de este experimento es comparar uno de los algoritmos que definen el estado del arte en cuanto a conjuntos de clasificadores se refiere, realizando una optimización de los hiperparámetros, con el algoritmo propuesto con los hiperparámetros también optimizados. Para realizar este experimento se ha escogido Random Forest como algoritmo representativo del estado del arte de los conjuntos de clasificadores por su sencillez a la hora de escoger los hiperparámetros a optimizar, por las similitudes que presenta con el modelo propuesto y, a su vez, por los notables

Tabla 4.3: Espacios de hiperparámetros utilizados.

Hiperparámetros	Valor por defecto	Rango de valores
Random forest		
`min_samples_split`	2	2 – 10
`max_depth`	None	2 – 15
`max_features`	'auto'	None, 'sqrt'
Redol		
`method`	'regular'	'regular', 'distributed'
`pil`	0.75	0.3 – 0.8
`bootstrap`	1.0	0.8 – 1.1
Redol (nearest neighbors)		
`nearest_neighbours`	None	4 – 12
`pil`	0.75	0.3 – 0.8

resultados que tiende a ofrecer este algoritmo independientemente de la variabilidad del problema que se presente.

La forma de llevar a cabo este experimento es la siguiente:

- Se escogen 10 divisiones aleatorias de cada conjunto de datos sobre el que se va a calcular la tasa de error. El experimento se va a realizar sobre todos los conjuntos de datos previamente descritos.
- Para cada división de los datos se escogen, de forma estratificada, un 66 % de las instancias para el subconjunto de entrenamiento y 33 % para el subconjunto de validación.
- Se establece el espacio de hiperparámetros sobre el que realizar la optimización. Para el método Redol se establecen dos espacios de búsqueda diferentes: uno con nearest_neighbors y otro sin nearest_neighbors. De esta forma, podemos comparar las diferentes opciones de este algoritmo entre sí. También se establece un espacio de hiperparámetros para el algoritmo Random Forest. Los espacios de hiperparámetros utilizados se pueden ver en la tabla 4.3.
- Finalmente, se realiza una búsqueda sobre los espacios de hiperparámetros para cada algoritmo y para cada iteración sobre el conjunto de datos. La búsqueda utilizada está basada en optimización bayesiana, por lo que no todos los parámetros del espacio se prueban, sino que se establece un número de iteraciones sobre las que esta búsqueda persigue el objetivo de obtener los parámetros que más ayuden a reducir el error de generalización del modelo. La búsqueda de hiperparámetros se realiza mediante una validación interna sobre el conjunto de entrenamiento y, una vez obtenidos los mejores parámetros para dicho subconjunto, se entrena el modelo y se calcula la tasa de error sobre el subconjunto de validación. El resultado final obtenido consiste en la media de los errores obtenidos para cada iteración.

El resultado final de este experimento está en la tabla 4.4. En la primera columna de esta tabla

Tabla 4.4: Comparación de los modelos hiperparametrizados. Se muestra la media del error de generalización junto a ± la desviación estándar. En **negrita** se resaltan los mejores resultados para cada conjunto de datos y subrayados se muestran los segundos mejores resultados.

Dataset	RF	RFOPT	REDOL	REDOLNN	REDOLOPT
wine	2.373±1.728	1.695±1.072	2.373±1.356	**1.356±1.478**	1.525±1.186
new-thyroid	5.07±2.687	**5.07±2.108**	7.606±2.76	5.352±2.873	6.197±3.461
heart	17.556±2.713	**17.111±2.685**	17.111±3.071	17.889±3.667	18.111±2.942
ionosphere	7.759±2.003	7.759±2.281	**6.466±1.601**	6.724±2.463	7.069±1.324
australian	**13.07±1.817**	13.202±1.365	13.596±1.787	13.86±1.712	13.947±2.027
wdbc	**4.628±1.142**	4.947±1.467	4.84±0.965	5.372±0.994	4.947±1.214
diabetes	24.094±2.162	24.016±1.633	**23.425±2.601**	23.898±1.985	23.543±2.732
tic-tac-toe	6.278±1.15	5.142±1.293	7.697±1.174	7.508±1.291	**4.732±1.602**
german	22.727±1.369	24.242±1.687	22.485±1.982	**22.394±1.577**	22.727±1.278
segment	2.477±0.728	2.569±0.765	3.329±0.398	**2.438±0.756**	2.739±0.593
magic04	**12.148±0.26**	12.356±0.293	12.334±0.226	12.332±0.242	12.285±0.287
twonorm	3.127±0.429	3.188±0.334	2.976±0.341	2.897±0.346	**2.855±0.32**
threenorm	14.43±0.701	14.188±0.747	13.648±0.608	**13.612±0.383**	13.685±0.59
ringnorm	4.418±0.365	4.806±0.398	3.491±0.297	3.388±0.317	**3.321±0.45**
waveform	14.867±0.67	14.97±0.633	**14.364±0.759**	14.582±0.734	14.406±0.747

se enumeran los conjuntos de datos sobre los que se ha realizado el experimento. Las dos siguientes columnas, RF y RFOPT, se corresponden con la media del error obtenido para el algoritmo Random Forest con los parámetros por defecto y con búsqueda de hiperparámetros, respectivamente. Las tres últimas columnas de la tabla son los resultados para el algoritmo REDOL con los parámetros por defecto, optimizando sus hiperparámetros con el método de vecinos cercanos y optimizando sus hiperparámetros con los métodos *regular* y *distributed*. Los valores de esta tabla se corresponden con la media del error obtenida para todas las iteraciones, para conjunto de datos y para cada algoritmo. Se indica también la desviación estándar para cada media de error. En negrita se marcan los mejores resultados obtenidos para cada conjunto de datos y subrayados, los segundos mejores resultados.

Utilizando la metodología descrita por (Demšar, 2006) podemos resumir los resultados obtenidos calculando el ranking medio para cada algoritmo sobre los conjuntos de datos probados. En la figura 4.4 podemos ver el resumen de los rankings obtenidos para cada algoritmo. En esta figura, un rango más bajo, implica un mejor rendimiento del modelo. No existen diferencias estadísticamente significantes entre los métodos que se encuentren conectados con líneas horizontales. Se considera estadísticamente significativo cuando se supera la distancia crítica indicada como CD en la figura. REDOL por defecto y con las dos hiperparametrizaciones distintas han obtenido en su conjunto mejores resultados que el algoritmo Random Forest tanto por defecto como con parámetros optimizados. Redol optimizado con vecinos cercanos es el que mejor ranking medio obtiene en comparación con el resto

de algoritmos. Sin embargo, no existen diferencias estadísticamente significativas entre los métodos comparados.

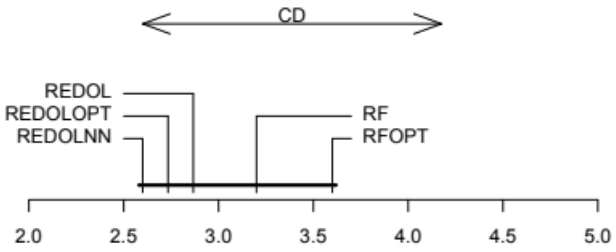


Figura 4.4: Comparativa de los rankings que obtienen los algoritmos con diferentes optimizaciones para todos los conjuntos de datos.

5

Resultados y discusión

Con los experimentos realizados se han podido analizar varios aspectos que influyen en las tomas de decisiones del modelo propuesto en este trabajo. Se ha centrado el análisis del modelo en los diferentes valores de aleatorización (parámetro pil), así como en otros parámetros que también tienen influencia en las decisiones finales como el número de árboles de decisión que componen el conjunto completo de clasificadores o los diferentes métodos de aleatorización que es posible utilizar para lograr que el algoritmo sea maleable dependiente al problema que se plantee.

Algunas observaciones destacables sobre el método y que se pueden extraer del análisis de los parámetros son: (1) la tendencia del algoritmo a la convergencia del error de generalización medio a medida que incrementa el número de clasificadores que se combinan; (2) la influencia que tiene el porcentaje de aleatorización en el entrenamiento del modelo; (3) el porcentaje de aleatorización (combinado con el método *regular*) ha demostrado comportarse de forma simétrica a partir de valores de 50 % para los distintos conjuntos de datos probados. Además, se han obtenido los mejores resultados para un amplio rango de valores dependientes del problema que se estuviese tratando. Es decir, para algunos problemas el porcentaje de aleatorización óptimo se podía centrar en 75 %, para otros en 50 % y, en general, cualquier valor entre el rango de 10 % – 90 % (teniendo en cuenta la simetría mencionada a partir de 50 %).

También se ha podido estudiar el rendimiento del algoritmo en comparación con los resultados de evaluar otros métodos de clasificación similares como Random Forest, Bagging y Gradient Boosting. El método propuesto, REDOL, con los parámetros por defecto es capaz de sobrepasar en rendimiento a varios de estos algoritmos, para 15 diversos conjuntos de datos. Entre los resultados obtenidos destacan tanto Random Forest como REDOL, consiguiendo entre ambos algoritmos los mejores resultados para 12 de los 15 conjuntos de datos. REDOL obtiene el menor error de generalización para 8 de los 15 *datasets* y Random Forest, para los 4 de los 15 conjuntos mencionados. Los resultados obtenidos con el algoritmo REDOL son remarcablemente buenos y competitivos para con el resto de algoritmos que definen el estado del arte del *ensemble learning*.

La evolución de la frontera de decisión nos permite observar cómo puede variar el algoritmo a medida que se aumenta el número de árboles de decisión combinados. Hemos podido observar cómo

la frontera de decisión tiende a suavizarse a medida que este número. Sin embargo, la frontera de decisión parece converger a partir de $\#estimators \approx 100$.

Finalmente, se han obtenido resultados sobresalientes en la comparación de REDOL con Random Forest buscando los hiperparámetros óptimos de cada algoritmo sobre un espacio de búsqueda similar. Para el algoritmo REDOL, en todas las variantes –por defecto, con búsqueda de hiperparámetros sobre vecinos cercanos y con búsqueda, sobre el resto de métodos– se han obtenido los errores más bajos de precisión para 9 de los 15 conjuntos de datos. Random Forest, por defecto y con optimización de hiperparámetros, ha obtenido los mejores resultados en 6 de los 15 conjuntos. Ordenando las posiciones obtenidas de cada algoritmos para cada *dataset* propuesto, teniendo en cuenta que el menor error de generalización sería la primera posición, obtenemos un ranking de los modelos para cada conjunto de datos. En la media de posiciones obtenidas para cada algoritmo hemos podido comprobar que el método propuesto, REDOL, obtiene las mejores posiciones en contraposición del algoritmo Random Forest.

Por estas razones podemos considerar que el algoritmo propuesto ha conseguido resultados realmente competitivos en comparación con otros algoritmos similares que definen el estado del arte en el reconocimiento de patrones por conjuntos de clasificadores, y se podrían considerar exitosos los objetivos propuestos en este trabajo.

6

Conclusiones

En este proyecto se ha propuesto un novedoso modelo basado en conjuntos de clasificadores que combina las predicciones que realizan una serie de árboles de decisión, sobre un conjunto de datos dado. El objetivo de este modelo es aprender a distinguir si una instancia está correctamente etiquetada o no, en contraste con los métodos estándares del aprendizaje automático que son entrenados para asignar una etiqueta a cada instancia. Para hacerlo, el método propuesto construye un conjunto de árboles de decisión entrenados en un problema diferente al original planteado. Cada árbol de decisión se entrena sobre un conjunto de datos dónde se añade una nueva variable, junto a las variables originales del problema, con una clase sugerida. El nuevo objetivo del problema en los que los modelos base son entrenados, es decidir si una instancia pertenece a una clase sugerida o no. Para construir un conjunto de clasificadores diverso, se generan diferentes versiones del problema original realizando muestras bootstrap y seleccionando una fracción –variable dependiente del método usado, pero generalmente interpretada como *pil*– de las instancias a las que se les asigna una clase sugerida errónea o diferente a la original. A las restantes instancias se les sugiere una clase correcta.

Cada clasificador combinado en el método propuesto es entrenado con un conjunto de datos modificado, de forma que aumente la diversidad del conjunto completo. La modificación propuesta consiste en la adición de una nueva variable en el conjunto de datos de entrenamiento. Esta nueva variable es una “clase sugerida“ para cada instancia del conjunto, de forma que la nueva etiqueta dada a este conjunto de datos es “clase sugerida correcta“ o “clase sugerida incorrecta“. Sin embargo, el método de selección de instancias que son modificadas y el método de modificación de las instancias para tener una clase errónea puede escogerse de entre cinco propuestos. En este trabajo se proponen cinco métodos diferentes: (*regular*) escoger aleatoriamente un número de instancias del conjunto dado por un porcentaje, *pil*; (*randomized*) asignarle a cada instancia de los datos una clase aleatoria de entre el total de etiquetas del conjunto, sin repetición. Esto equivale a barajar las etiquetas de los datos entre las instancias; (*distributed*) escoger las instancias de forma que al modificar las etiquetas se mantenga la distribución original del conjunto de datos; (*nearest neighbors*) asignar un porcentaje *pil* de instancias a modificar y *k* instancias vecinas para cada instancia, de forma que a cada instancia seleccionada es modificada con la etiqueta de uno de los datos con más similitud; (*no pil*) similar al método de *nearest neighbors*, con la distinción de que en este caso no se utiliza *pil* para la selección de instancias, sino

que sólo se modifican aquellas instancias cuyos vecinos cercanos tengan una etiqueta distinta a la original del dato.

El método propuesto aplicado a la regresión ha sido probado y estudiado en este proyecto. Con un concepto similar a la aleatorización de las instancias añadiendo un nuevo atributo de una clase sugerida, se ha implementado el mismo método propuesto, pero con el objetivo de resolver problemas de regresión. Sin embargo, con este método no se han obtenido los resultados esperados y se ha descartado la continuación de su análisis y desarrollo.

El algoritmo en este proyecto propuesto se ha evaluado sobre una serie de experimentos para valorar la eficacia de las predicciones que realiza. En estos experimentos se ha estudiado y analizado la evolución del algoritmo al modificar los parámetros de aleatorización y del número de árboles combinados, la diversidad de los clasificadores base que se combinan, la precisión obtenida en comparación con otros algoritmos similares de clasificación basados en la combinación de clasificadores más simples, la frontera de decisión generada y la evolución de la misma al modificar el número de árboles de decisión implicados en el conjunto y, finalmente, la precisión obtenida al optimizar los hiperparámetros del modelo y la comparación con el algoritmo Random Forest, también con optimización de sus hiperparámetros. Los experimentos se han realizado sobre una serie de 15 conjuntos de datos diversos en distribución de clases, número de variables y carga total de instancias. Con el análisis de los resultados obtenidos en los experimentos realizados se han podido señalar una serie de observaciones descriptivas del rendimiento final del modelo. Como ocurre con otros métodos basados en conjuntos de clasificadores, a medida que el número de clasificadores que se combinan aumenta, el error de generalización se estabiliza. Otra observación de interés es que los mejores valores de precisión conseguidos se alcanzan para un amplio rango de valores para el porcentaje de aleatorización pil, $pil \approx [10\,\%, 90\,\%]$, dependiente del problema que se quiera resolver. La frontera de decisión se suaviza al combinar un mayor número de clasificadores base en el modelo, aunque tiende a converger al combinar 100 árboles de decisión. Finalmente, el modelo propuesto ha sido comparado con Random Forest y otros métodos similares, consiguiendo mejores o equivalentes resultados en los conjuntos de datos analizados. Es por esto que se pueden considerar exitosos los experimentos realizados y los resultados obtenidos han resultado ser realmente competitivos en comparación con algoritmos destacables y frecuentemente utilizados en el ámbito del reconocimiento de patrones.

6.1. Trabajo futuro

Aunque se han completado todos los objetivos propuestos en este proyecto, el desarrollo e investigación sobre el modelo planteado en esta memoria aún puede evolucionar más. En este informe se han planteado, no sólo el método, sino modificaciones del mismo para optimizar su rendimiento, y experimentos que nos ayudan a comprender tanto el comportamiento interno del modelo como la eficacia del algoritmo en problemas reales. El modelo admite más desarrollo en estos aspectos e incluso

ofrece la posibilidad de una mejora continua, es por esto que se plantean en este apartado una serie de puntos para continuar con el desarrollo del modelo:

- Todos los parámetros implementados han implicado cambios en el entrenamiento del algoritmo, sin embargo, pueden existir variaciones en cuanto a la clasificación. Por ejemplo, para clasificar se utiliza como clase sugerida todas las etiquetas posibles del conjunto de datos, lo cual lleva a una pérdida de eficiencia en clasificación que aumenta con el número de etiquetas. Una opción sería utilizar un número de iteraciones sobre las que aleatorizar la clase sugerida y realizar la media de los resultados.
- Los experimentos se han realizado sobre un rango concreto de valores para el método de aleatorización y para el porcentaje *pil* de selección de instancias. Sería ideal realizar una nueva serie de experimentos donde se analice el comportamiento del modelo sobre el resto de valores posibles para los parámetros descritos.
- Debido al elevado coste computacional que supone realizar los experimentos en este proyecto propuestos, el número total de conjuntos de datos utilizados se ha visto limitado por esta restricción. Sería idea poder probar todos los experimentos sobre una batería de datos mucho más completa, sobre todo en términos de tamaño –número total de instancias–, pero también en términos de diversidad de número de atributos y número de etiquetas.
- El desarrollo de los distintos parámetros se ha visto limitado por las restricciones temporales de este proyecto, sin embargo, las posibilidades de optimización y desarrollo del algoritmos en base a sus atributos es todavía muy amplia. Se podrían considerar nuevos métodos de aleatorización y optimización del modelo propuesto para poder adaptarlo mejor a los distintos problemas que se propongan.

BIBLIOGRAFÍA

C. Aggarwal and C. Zhai. *A survey of text classification algorithms*, volume 9781461432234. 2012. doi: 10.1007/978-1-4614-3223-4_6.

L. Alzubaidi, J. Zhang, A. Humaidi, A. Al-Dujaili, Y. Duan, O. Al-Shamma, J. Santamaría, M. Fadhel, M. Al-Amidie, and L. Farhan. Review of deep learning: concepts, cnn architectures, challenges, applications, future directions. *Journal of Big Data*, 8(1), 2021. doi: 10.1186/s40537-021-00444-8.

C. Bentéjac, A. Csörgő, and G. Martínez-Muñoz. A comparative analysis of gradient boosting algorithms. *Artificial Intelligence Review*, 54(3):1937–1967, 2021. doi: 10.1007/s10462-020-09896-5.

C. Bishop. *Pattern Recognition and Machine Learning*, volume 16, pages 140–155. 01 2006. doi: 10.1117/1.2819119.

L. Breiman. Bias, variance, and arcing classifiers. Technical report, 1996a.

L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996b.

L. Breiman. Randomizing outputs to increase prediction accuracy. *Machine Learning*, 40(3):229–242, 2000.

L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and regression trees*. 2017. doi: 10.1201/9781315139470.

T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4232-2.

J. Demšar. Statistical comparisons of classifiers over multiple data sets. *J. Mach. Learn. Res.*, 7:1–30, Dec. 2006. ISSN 1532-4435.

T. G. Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning*, 40(2):139–157, 2000.

D. Dua and C. Graff. UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`.

M. Fernández-Delgado, E. Cernadas, S. Barro, and D. Amorim. Do we need hundreds of classifiers to solve real world classification problems? *Journal of Machine Learning Research*, (15):3133–3181, 2014.

Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, Aug. 1997.

J. H. Friedman. Greedy function approximation: a Gradient Boosting machine. *The Annals of Statistics*, 29(5):1189 – 1232, 2001.

M. Gupta and G. Batra. Investigation of machine learning assistance to education. pages 777–782, 2021. doi: 10.1109/ICCMC51019.2021.9418364.

E. B. Kong and T. G. Dietterich. Error-correcting output coding corrects bias and variance. In A. Prieditis and S. Russell, editors, *Machine Learning Proceedings 1995*, pages 313–321. Morgan Kaufmann, San Francisco (CA), 1995. ISBN 978-1-55860-377-6. doi: https://doi.org/10.1016/B978-1-55860-377-6.50046-3.

G. Martínez-Muñoz, A. Sánchez-Martínez, D. Hernández-Lobato, and A. Suárez. Class-switching neural network ensembles. *Neurocomputing*, 13–15(71):2521–2528, 2008.

G. Martínez-Muñoz and A. Suárez. Switching class labels to generate classification ensembles. *Pattern Recognition*, 38(10):1483–1494, 2005.

M. Nemesure, M. Heinz, R. Huang, and N. Jacobson. Predictive modeling of depression and anxiety using electronic health records and a novel machine learning approach with artificial intelligence. *Scientific Reports*, 11(1), 2021. doi: 10.1038/s41598-021-81368-4. cited By 1.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

G. Preud'homme, K. Duarte, K. Dalleau, C. Lacomblez, E. Bresso, M. Smaïl-Tabbone, M. Couceiro, M.-D. Devignes, M. Kobayashi, O. Huttin, J. Ferreira, F. Zannad, P. Rossignol, and N. Girerd. Head-to-head comparison of clustering methods for heterogeneous data: a simulation-driven benchmark. *Scientific Reports*, 11(1), 2021. doi: 10.1038/s41598-021-83340-8.

L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin. Catboost: Unbiased boosting with categorical features. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 6639–6649, Red Hook, NY, USA, 2018. Curran Associates Inc.

M. Ravaut, H. Sadeghi, K. Leung, M. Volkovs, K. Kornas, V. Harish, T. Watson, G. Lewis, A. Weisman, T. Poutanen, and L. Rosella. Predicting adverse outcomes due to diabetes complications with machine learning using administrative health data. *npj Digital Medicine*, 4(1), 2021. doi: 10.1038/s41746-021-00394-8.

R. Schapire, Y. Freund, P. Bartlett, and W. Lee. Boosting the margin: A new explanation for the effectiveness of voting methods. *Annals of Statistics*, 26(5):1651–1686, 1998. doi: 10.1214/aos/1024691352.

M. Smith, L. Smith, and M. Hansen. The quiet revolution in machine vision - a state-of-the-art survey paper, including historical review, perspectives, and future directions. *Computers in Industry*, 130, 2021. doi: 10.1016/j.compind.2021.103472.

E. Tacchini, G. Ballarin, M. L. D. Vedova, S. Moret, and L. de Alfaro. Some like it hoax: Automated fake news detection in social networks, 2017.

S. Wang, Y. Zha, W. Li, Q. Wu, X. Li, M. Niu, M. Wang, X. Qiu, H. Li, H. Yu, W. Gong, Y. Bai, L. Li, Y. Zhu, L. Wang, and J. Tian. A fully automatic deep learning system for covid-19 diagnostic and prognostic analysis. *European Respiratory Journal*, 56(2), 2020. doi: 10.1183/13993003.00775-2020.

C. Zhang, C. Liu, X. Zhang, and G. Almpanidis. An up-to-date comparison of state-of-the-art classification algorithms. *Expert Systems with Applications*, 82:128–150, 2017.